Basi di dati II — Prova parziale — 12 maggio 2015 — Compito A

Tempo a disposizione: un'ora e trenta minuti.

Si suggerisce di scrivere prima una brutta copia, per indicare poi negli spazi le risposte e brevi giustificazioni.

Cognome _____ **Nome** _____ **Matricola** _____

Domanda 1 (20%)

Considerare una tabella `T` appena creata (e quindi vuota), con le seguenti ipotesi

- `T` è definita su due campi, `A` di lunghezza $a = 4$ byte e `B` di lunghezza $b = 34$ byte, senza vincoli espliciti di chiave (e quindi le operazioni si possono fare senza verifiche particolari);
- la struttura fisica utilizzata per `T` è heap, senza indici, con una memorizzazione a lunghezza fissa (in cui supponiamo che, oltre ai byte necessari per i campi, ne servano 2 ulteriori per la memorizzazione) e in cui si marcano come liberi gli spazi dei record eliminati, riutilizzandoli per successivi inserimenti (come avviene in SimpleDB);
- il sistema utilizza blocchi di dimensione $D = 4$ Kbyte (approssimabili a 4000).

In tale contesto, supporre che vengano eseguite le seguenti operazioni

1. inserimento di $L = 200.000$ ennuple
2. eliminazione di $L/4 = 50.000$ ennuple (sulla base di una condizione verificabile durante la scansione)
3. dopo la conclusione e la chiusura della scansione precedente, inserimento di altre $L/4 = 50.000$ ennuple

Rispondere alle domande seguenti, indicando formule e valori numerici:

Fattore di blocco f per la relazione `T`:

Numero blocchi occupati da `T` dopo la prima serie di inserimenti (punto 1):

Numero blocchi occupati da `T` dopo le eliminazioni di cui al punto 2:

Numero blocchi occupati da `T` dopo la seconda serie di inserimenti (punto 3):

Numero di scritture in memoria secondaria (per le pagine dei dati e quelle del log) per la prima serie di inserimenti, supponendo che, in questo caso, i record di log abbiano una lunghezza pari a circa il doppio di quella dei record del file, con riferimento ad un programma che utilizzi un'unica transazione per tutti gli inserimenti (e supponendo che non vi siano altre transazioni attive)

- numero di scritture di pagine di log:
- numero di scritture di pagine di dati:

Come nel caso precedente, ma con riferimento ad un programma che utilizzi una transazione separata per ciascun inserimento

- numero di scritture di pagine di log:
- numero di scritture di pagine di dati:

Basi di dati II — 12 maggio 2015 — Compito A

Domanda 2 (15%)
Come noto, esistono tecniche leggermente diverse per il recovery, che prevedono solo redo (**"redo-only"**), solo undo (**"undo-only"**) oppure tanto undo quanto redo (**"undo-redo"**). Le tecniche differiscono, oltre che nella procedura di recovery, anche nella modalità di esecuzione delle scritture effettive su disco (operazioni di "flush" delle pagine di buffer), che possono essere le seguenti (con riferimento ad un utilizzo di checkpoint non quiescente):

flush-nel-commit le pagine modificate da una transazione vengono scritte su disco (subito) prima del commit della transazione stessa;

flush-all-nel-ckpt nel corso del checkpoint (subito prima della sua conclusione), vengono scritte su disco le pagine modificate da tutte le transazioni;

flush-committed-nel-ckpt nel corso del checkpoint (subito prima della sua conclusione), vengono scritte su disco le pagine modificate dalle transazioni andate in commit.

Nella tabella seguente, indicare per ciascuna tecnica quali approcci alla flush possono o debbono essere utilizzati (indicare il nome sintetico e fornire una breve spiegazione; si noti che potrebbe essere utilizzabile una sola tecnica oppure più di una; in questo secondo caso, indicare e spiegare quale sarebbe preferibile).

1. **"undo-redo"**

2. **"redo-only"**

3. **"undo-only"**

Basi di dati II — 12 maggio 2015 — Compito A

Domanda 3 (15%)
Si consideri una base di dati su cui una applicazione effettua moltissimi inserimenti e aggiornamenti, con operazioni tutte molto semplici ma estremamente numerose. Considerare le due situazioni seguenti:

A. concorrentemente alle operazioni sopra citate non viene eseguita nessun'altra operazione
B. concorrentemente alle operazioni sopra citate vengono eseguite molte interrogazioni (e nessun altro aggiornamento)

Commentare brevemente per ciascuna delle due situazioni quali potrebbero essere i vantaggi e gli svantaggi (con riferimento alle prestazioni in termini di tempo di risposta sia per la normale operatività sia in caso di guasto e conseguente necessità di recovery) delle seguenti tre scelte per l'applicazione che esegue inserimenti e aggiornamenti:

i. eseguire ciascuna operazione in una transazione separata
ii. riunire le operazioni in transazioni di medie dimensioni (alcune decine di operazioni per ciascuna)
iii. riunire tutte le operazioni in un'unica transazione

Nota bene: non è detto che esista una soluzione ideale, ciòò che si deve fornire sono riflessioni critiche, che, sinteticamente, illustrino spunti interessanti.

	A	B
i.		
ii.		
iii.		

Basi di dati II — 12 maggio 2015 — Compito A

Domanda 4 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
		begin
		read(x)
begin		
read(x)		
	begin	
	read(x)	
x = x + 10		
write(x)		
commit		
	x = x + 20	
		read(x)
	write(x)	
	commit	
		commit

Considerare uno scheduler con controllo di concorrenza basato su **2PL** e livello di isolamento **READ COMMITTED** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 400. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — 12 maggio 2015 — Compito A

Domanda 5 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
begin		
read(x)		
	begin	
	read(x)	
x = x + 10		
write(x)		
		begin
		read(x)
	x = x + 20	
	write(x)	
	commit	
commit		
		read(x)
		commit

Considerare uno scheduler con controllo di concorrenza basato su **Multiversioni** e livello di isolamento **SERIALIZABLE** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 400. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3

Si verificano anomalie?

Indicare brevemente che cosa succede se invece la transazione sul client 3 ha livello di isolamento READ COMMITTED.

Basi di dati II — 12 maggio 2015 — Compito A

Domanda 6 (20%)
Considerare un sistema distribuito su cui vengono eseguite due transazioni che coinvolgono tre nodi, un coordinatore M (lo stesso per entrambe le transazioni) e due partecipanti R1 e R2. Dopo la richiesta del coordinatore M di `prepare` (abbreviata con `prep`) per la prima transazione, i due partecipanti ricevono e rispondono correttamente, e uno dei due, R2, va in crash subito dopo aver risposto. Il coordinatore riceve le risposte, prende la decisione, invia i relativi messaggi (questi passi **non** sono stati indicati sotto e vanno quindi scritti) e subito dopo va anch'esso in crash (quindi senza fare in tempo a ricevere le conferme). Viceversa, per la seconda transazione, il coordinatore invia il messaggio di `prepare` ma non fa in tempo a ricevere le risposte. Indicare, nello schema sottostante, una possibile sequenza di scritture sui log e invio di messaggi (che includa anche i passi sopra illustrati), supponendo che entrambi i nodi siano ripristinati abbastanza presto (ma che il coordinatore perda alcuni messaggi di risposta inviati ad esso a seguito del commit). Per i messaggi si usi la notazione *tipo(transaz)*→*destinatari* (come nell'esempio: `prepare(T1)`→`R1,R2`). Supporre che nel log del coordinatore si scrivano solo i record di `prepare`, `commit` e `complete`, con i messaggi gestiti invece in memoria. Indicare ragionevoli istanti per i timeout, che permettano di concludere entrambe le transazioni.

Nodo M		Nodo R1		Nodo R2	
Log	Messaggi	Log	Messaggi	Log	Messaggi
`prep(T1,R1,R2)`	`prep(T1)`→`R1,R2`				
					crash
`prep(T2,R1,R2)`	`prep(T2)`→`R1,R2`				
	crash				
	restart				
					restart

Basi di dati II — Prova parziale — 12 maggio 2015 — Compito B

Tempo a disposizione: un'ora e trenta minuti.

Si suggerisce di scrivere prima una brutta copia, per indicare poi negli spazi le risposte e brevi giustificazioni.

Cognome _____ **Nome** _____ **Matricola** _____

Domanda 1 (20%)

Considerare una tabella `T` appena creata (e quindi vuota), con le seguenti ipotesi

- `T` è definita su due campi, `A` di lunghezza $a = 6$ byte e `B` di lunghezza $b = 72$ byte, senza vincoli espliciti di chiave (e quindi le operazioni si possono fare senza verifiche particolari);
- la struttura fisica utilizzata per `T` è heap, senza indici, con una memorizzazione a lunghezza fissa (in cui supponiamo che, oltre ai byte necessari per i campi, ne servano 2 ulteriori per la memorizzazione) e in cui si marcano come liberi gli spazi dei record eliminati, riutilizzandoli per successivi inserimenti (come avviene in SimpleDB);
- il sistema utilizza blocchi di dimensione $D = 8$ Kbyte (approssimabili a 8000).

In tale contesto, supporre che vengano eseguite le seguenti operazioni

1. inserimento di $N = 200.000$ ennuple
2. eliminazione di $N/4 = 50.000$ ennuple (sulla base di una condizione verificabile durante la scansione)
3. dopo la conclusione e la chiusura della scansione precedente, inserimento di altre $N/4 = 50.000$ ennuple

Rispondere alle domande seguenti, indicando formule e valori numerici:

Fattore di blocco f per la relazione `T`:

Numero blocchi occupati da `T` dopo la prima serie di inserimenti (punto 1):

Numero blocchi occupati da `T` dopo le eliminazioni di cui al punto 2:

Numero blocchi occupati da `T` dopo la seconda serie di inserimenti (punto 3):

Numero di scritture in memoria secondaria (per le pagine dei dati e quelle del log) per la prima serie di inserimenti, supponendo che, in questo caso, i record di log abbiano una lunghezza pari a circa il doppio di quella dei record del file, con riferimento ad un programma che utilizzi un'unica transazione per tutti gli inserimenti (e supponendo che non vi siano altre transazioni attive)

- numero di scritture di pagine di log:
- numero di scritture di pagine di dati:

Come nel caso precedente, ma con riferimento ad un programma che utilizzi una transazione separata per ciascun inserimento

- numero di scritture di pagine di log:
- numero di scritture di pagine di dati:

Basi di dati II — 12 maggio 2015 — Compito B

Domanda 2 (15%)
Come noto, esistono tecniche leggermente diverse per il recovery, che prevedono solo redo (**"redo-only"**), solo undo (**"undo-only"**) oppure tanto undo quanto redo (**"undo-redo"**). Le tecniche differiscono, oltre che nella procedura di recovery, anche nella modalità di esecuzione delle scritture effettive su disco (operazioni di "flush" delle pagine di buffer), che possono essere le seguenti (con riferimento ad un utilizzo di checkpoint non quiescente):

flush-nel-commit le pagine modificate da una transazione vengono scritte su disco (subito) prima del commit della transazione stessa;

flush-all-nel-ckpt nel corso del checkpoint (subito prima della sua conclusione), vengono scritte su disco le pagine modificate da tutte le transazioni;

flush-committed-nel-ckpt nel corso del checkpoint (subito prima della sua conclusione), vengono scritte su disco le pagine modificate dalle transazioni andate in commit.

Nella tabella seguente, indicare per ciascuna tecnica quali approcci alla flush possono o debbono essere utilizzati (indicare il nome sintetico e fornire una breve spiegazione; si noti che potrebbe essere utilizzabile una sola tecnica oppure più di una; in questo secondo caso, indicare e spiegare quale sarebbe preferibile).

1. **"redo-only"**

2. **"undo-only"**

3. **"undo-redo"**

Basi di dati II — 12 maggio 2015 — Compito B

Domanda 3 (15%)
Si consideri una base di dati su cui una applicazione effettua moltissimi inserimenti e aggiornamenti, con operazioni tutte molto semplici ma estremamente numerose. Considerare le due situazioni seguenti:

A. concorrentemente alle operazioni sopra citate vengono eseguite molte interrogazioni (e nessun altro aggiornamento)
B. concorrentemente alle operazioni sopra citate non viene eseguita nessun'altra operazione

Commentare brevemente per ciascuna delle due situazioni quali potrebbero essere i vantaggi e gli svantaggi (con riferimento alle prestazioni in termini di tempo di risposta sia per la normale operatività sia in caso di guasto e conseguente necessità di recovery) delle seguenti tre scelte per l'applicazione che esegue inserimenti e aggiornamenti:

i. eseguire ciascuna operazione in una transazione separata
ii. riunire tutte le operazioni in un'unica transazione
iii. riunire le operazioni in transazioni di medie dimensioni (alcune decine di operazioni per ciascuna)

Nota bene: non è detto che esista una soluzione ideale, ciòò che si deve fornire sono riflessioni critiche, che, sinteticamente, illustrino spunti interessanti.

	A	B
i.		
ii.		
iii.		

Basi di dati II — 12 maggio 2015 — Compito B

Domanda 4 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
begin		
read(x)		
	begin	
	read(x)	
x = x + 10		
write(x)		
		begin
		read(x)
	x = x + 20	
	write(x)	
	commit	
commit		
		read(x)
		commit

Considerare uno scheduler con controllo di concorrenza basato su **2PL** e livello di isolamento **SERIALIZABLE** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 400. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3

Si verificano anomalie?

Indicare brevemente che cosa succede se invece la transazione sul client 3 ha livello di isolamento READ COMMITTED.

Basi di dati II — 12 maggio 2015 — Compito B

Domanda 5 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
		begin
		read(x)
begin		
read(x)		
	begin	
	read(x)	
x = x + 10		
write(x)		
commit		
	x = x + 20	
		read(x)
	write(x)	
	commit	
		commit

Considerare uno scheduler con controllo di concorrenza basato su **Multiversioni** e livello di isolamento **READ COMMITTED** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 400. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — 12 maggio 2015 — Compito B

Domanda 6 (20%)
Considerare un sistema distribuito su cui vengono eseguite due transazioni che coinvolgono tre nodi, un coordinatore C (lo stesso per entrambe le transazioni) e due partecipanti N1 e N2. Dopo la richiesta del coordinatore C di `prepare` (abbreviata con `prep`) per la prima transazione, i due partecipanti ricevono e rispondono correttamente, e uno dei due, N2, va in crash subito dopo aver risposto. Il coordinatore riceve le risposte, prende la decisione, invia i relativi messaggi (questi passi **non** sono stati indicati sotto e vanno quindi scritti) e subito dopo va anch'esso in crash (quindi senza fare in tempo a ricevere le conferme). Viceversa, per la seconda transazione, il coordinatore invia il messaggio di `prepare` ma non fa in tempo a ricevere le risposte. Indicare, nello schema sottostante, una possibile sequenza di scritture sui log e invio di messaggi (che includa anche i passi sopra illustrati), supponendo che entrambi i nodi siano ripristinati abbastanza presto (ma che il coordinatore perda alcuni messaggi di risposta inviati ad esso a seguito del commit). Per i messaggi si usi la notazione *tipo(transaz)*→*destinatari* (come nell'esempio: `prepare(T1)`→`N1,N2`). Supporre che nel log del coordinatore si scrivano solo i record di `prepare`, `commit` e `complete`, con i messaggi gestiti invece in memoria. Indicare ragionevoli istanti per i timeout, che permettano di concludere entrambe le transazioni.

Nodo C		Nodo N1		Nodo N2	
Log	Messaggi	Log	Messaggi	Log	Messaggi
`prep(T1,N1,N2)`	`prep(T1)`→`N1,N2`				
					crash
`prep(T2,N1,N2)`	`prep(T2)`→`N1,N2`				
	crash				
	restart				
					restart

Basi di dati II — Prova parziale — 12 maggio 2015 — Compito C

Tempo a disposizione: un'ora e trenta minuti.

Si suggerisce di scrivere prima una brutta copia, per indicare poi negli spazi le risposte e brevi giustificazioni.

Cognome _____ **Nome** _____ **Matricola** _____

Domanda 1 (20%)

Considerare una tabella `R` appena creata (e quindi vuota), con le seguenti ipotesi

- `R` è definita su due campi, `A` di lunghezza $a = 4$ byte e `B` di lunghezza $b = 34$ byte, senza vincoli espliciti di chiave (e quindi le operazioni si possono fare senza verifiche particolari);
- la struttura fisica utilizzata per `R` è heap, senza indici, con una memorizzazione a lunghezza fissa (in cui supponiamo che, oltre ai byte necessari per i campi, ne servano 2 ulteriori per la memorizzazione) e in cui si marcano come liberi gli spazi dei record eliminati, riutilizzandoli per successivi inserimenti (come avviene in SimpleDB);
- il sistema utilizza blocchi di dimensione $D = 4$ Kbyte (approssimabili a 4000).

In tale contesto, supporre che vengano eseguite le seguenti operazioni

1. inserimento di $L = 200.000$ ennuple
2. eliminazione di $L/4 = 50.000$ ennuple (sulla base di una condizione verificabile durante la scansione)
3. dopo la conclusione e la chiusura della scansione precedente, inserimento di altre $L/4 = 50.000$ ennuple

Rispondere alle domande seguenti, indicando formule e valori numerici:

Fattore di blocco f per la relazione `R`:

Numero blocchi occupati da `R` dopo la prima serie di inserimenti (punto 1):

Numero blocchi occupati da `R` dopo le eliminazioni di cui al punto 2:

Numero blocchi occupati da `R` dopo la seconda serie di inserimenti (punto 3):

Numero di scritture in memoria secondaria (per le pagine dei dati e quelle del log) per la prima serie di inserimenti, supponendo che, in questo caso, i record di log abbiano una lunghezza pari a circa il doppio di quella dei record del file, con riferimento ad un programma che utilizzi un'unica transazione per tutti gli inserimenti (e supponendo che non vi siano altre transazioni attive)

- numero di scritture di pagine di log:
- numero di scritture di pagine di dati:

Come nel caso precedente, ma con riferimento ad un programma che utilizzi una transazione separata per ciascun inserimento

- numero di scritture di pagine di log:
- numero di scritture di pagine di dati:

Basi di dati II — 12 maggio 2015 — Compito C

Domanda 2 (15%)
Come noto, esistono tecniche leggermente diverse per il recovery, che prevedono solo redo (**"redo-only"**), solo undo (**"undo-only"**) oppure tanto undo quanto redo (**"undo-redo"**). Le tecniche differiscono, oltre che nella procedura di recovery, anche nella modalità di esecuzione delle scritture effettive su disco (operazioni di "flush" delle pagine di buffer), che possono essere le seguenti (con riferimento ad un utilizzo di checkpoint non quiescente):

flush-nel-commit le pagine modificate da una transazione vengono scritte su disco (subito) prima del commit della transazione stessa;

flush-all-nel-ckpt nel corso del checkpoint (subito prima della sua conclusione), vengono scritte su disco le pagine modificate da tutte le transazioni;

flush-committed-nel-ckpt nel corso del checkpoint (subito prima della sua conclusione), vengono scritte su disco le pagine modificate dalle transazioni andate in commit.

Nella tabella seguente, indicare per ciascuna tecnica quali approcci alla flush possono o debbono essere utilizzati (indicare il nome sintetico e fornire una breve spiegazione; si noti che potrebbe essere utilizzabile una sola tecnica oppure più di una; in questo secondo caso, indicare e spiegare quale sarebbe preferibile).

1. **"undo-redo"**

2. **"redo-only"**

3. **"undo-only"**

Basi di dati II — 12 maggio 2015 — Compito C

Domanda 3 (15%)
Si consideri una base di dati su cui una applicazione effettua moltissimi inserimenti e aggiornamenti, con operazioni tutte molto semplici ma estremamente numerose. Considerare le due situazioni seguenti:

A. concorrentemente alle operazioni sopra citate non viene eseguita nessun'altra operazione
B. concorrentemente alle operazioni sopra citate vengono eseguite molte interrogazioni (e nessun altro aggiornamento)

Commentare brevemente per ciascuna delle due situazioni quali potrebbero essere i vantaggi e gli svantaggi (con riferimento alle prestazioni in termini di tempo di risposta sia per la normale operatività sia in caso di guasto e conseguente necessità di recovery) delle seguenti tre scelte per l'applicazione che esegue inserimenti e aggiornamenti:

i. riunire tutte le operazioni in un'unica transazione
ii. eseguire ciascuna operazione in una transazione separata
iii. riunire le operazioni in transazioni di medie dimensioni (alcune decine di operazioni per ciascuna)

Nota bene: non è detto che esista una soluzione ideale, ciòò che si deve fornire sono riflessioni critiche, che, sinteticamente, illustrino spunti interessanti.

	A	B
i.		
ii.		
iii.		

Basi di dati II — 12 maggio 2015 — Compito C

Domanda 4 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
begin		
read(x)		
	begin	
	read(x)	
x = x + 10		
write(x)		
		begin
		read(x)
	x = x + 20	
	write(x)	
	commit	
commit		
		read(x)
		commit

Considerare uno scheduler con controllo di concorrenza basato su **2PL** e livello di isolamento **READ COMMITTED** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 400. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — 12 maggio 2015 — Compito C

Domanda 5 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
		begin
		read(x)
begin		
read(x)		
	begin	
	read(x)	
x = x + 10		
write(x)		
commit		
	x = x + 20	
		read(x)
	write(x)	
	commit	
		commit

Considerare uno scheduler con controllo di concorrenza basato su **Multiversioni** e livello di isolamento **SERIALIZABLE** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 400. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3

Si verificano anomalie?

Indicare brevemente che cosa succede se invece la transazione sul client 3 ha livello di isolamento READ COMMITTED.

Basi di dati II — 12 maggio 2015 — Compito C

Domanda 6 (20%)
Considerare un sistema distribuito su cui vengono eseguite due transazioni che coinvolgono tre nodi, un coordinatore M (lo stesso per entrambe le transazioni) e due partecipanti R1 e R2. Dopo la richiesta del coordinatore M di `prepare` (abbreviata con `prep`) per la prima transazione, i due partecipanti ricevono e rispondono correttamente, e uno dei due, R2, va in crash subito dopo aver risposto. Il coordinatore riceve le risposte, prende la decisione, invia i relativi messaggi (questi passi **non** sono stati indicati sotto e vanno quindi scritti) e subito dopo va anch'esso in crash (quindi senza fare in tempo a ricevere le conferme). Viceversa, per la seconda transazione, il coordinatore invia il messaggio di `prepare` ma non fa in tempo a ricevere le risposte. Indicare, nello schema sottostante, una possibile sequenza di scritture sui log e invio di messaggi (che includa anche i passi sopra illustrati), supponendo che entrambi i nodi siano ripristinati abbastanza presto (ma che il coordinatore perda alcuni messaggi di risposta inviati ad esso a seguito del commit). Per i messaggi si usi la notazione *tipo(transaz)*→*destinatari* (come nell'esempio: `prepare(T1)`→`R1,R2`). Supporre che nel log del coordinatore si scrivano solo i record di `prepare`, `commit` e `complete`, con i messaggi gestiti invece in memoria. Indicare ragionevoli istanti per i timeout, che permettano di concludere entrambe le transazioni.

Nodo M		Nodo R1		Nodo R2	
Log	Messaggi	Log	Messaggi	Log	Messaggi
`prep(T1,R1,R2)`	`prep(T1)`→`R1,R2`				
					crash
`prep(T2,R1,R2)`	`prep(T2)`→`R1,R2`				
	crash				
	restart				
					restart

Basi di dati II — Prova parziale — 12 maggio 2015 — Compito D

Tempo a disposizione: un'ora e trenta minuti.

Si suggerisce di scrivere prima una brutta copia, per indicare poi negli spazi le risposte e brevi giustificazioni.

Cognome _____ **Nome** _____ **Matricola** _____

Domanda 1 (20%)

Considerare una tabella `R` appena creata (e quindi vuota), con le seguenti ipotesi

- `R` è definita su due campi, `A` di lunghezza $a = 6$ byte e `B` di lunghezza $b = 72$ byte, senza vincoli espliciti di chiave (e quindi le operazioni si possono fare senza verifiche particolari);
- la struttura fisica utilizzata per `R` è heap, senza indici, con una memorizzazione a lunghezza fissa (in cui supponiamo che, oltre ai byte necessari per i campi, ne servano 2 ulteriori per la memorizzazione) e in cui si marcano come liberi gli spazi dei record eliminati, riutilizzandoli per successivi inserimenti (come avviene in SimpleDB);
- il sistema utilizza blocchi di dimensione $D = 8$ Kbyte (approssimabili a 8000).

In tale contesto, supporre che vengano eseguite le seguenti operazioni

1. inserimento di $N = 200.000$ ennuple
2. eliminazione di $N/4 = 50.000$ ennuple (sulla base di una condizione verificabile durante la scansione)
3. dopo la conclusione e la chiusura della scansione precedente, inserimento di altre $N/4 = 50.000$ ennuple

Rispondere alle domande seguenti, indicando formule e valori numerici:

Fattore di blocco f per la relazione `R`:

Numero blocchi occupati da `R` dopo la prima serie di inserimenti (punto 1):

Numero blocchi occupati da `R` dopo le eliminazioni di cui al punto 2:

Numero blocchi occupati da `R` dopo la seconda serie di inserimenti (punto 3):

Numero di scritture in memoria secondaria (per le pagine dei dati e quelle del log) per la prima serie di inserimenti, supponendo che, in questo caso, i record di log abbiano una lunghezza pari a circa il doppio di quella dei record del file, con riferimento ad un programma che utilizzi un'unica transazione per tutti gli inserimenti (e supponendo che non vi siano altre transazioni attive)

- numero di scritture di pagine di log:
- numero di scritture di pagine di dati:

Come nel caso precedente, ma con riferimento ad un programma che utilizzi una transazione separata per ciascun inserimento

- numero di scritture di pagine di log:
- numero di scritture di pagine di dati:

Basi di dati II — 12 maggio 2015 — Compito D

Domanda 2 (15%)
Come noto, esistono tecniche leggermente diverse per il recovery, che prevedono solo redo (**"redo-only"**), solo undo (**"undo-only"**) oppure tanto undo quanto redo (**"undo-redo"**). Le tecniche differiscono, oltre che nella procedura di recovery, anche nella modalità di esecuzione delle scritture effettive su disco (operazioni di "flush" delle pagine di buffer), che possono essere le seguenti (con riferimento ad un utilizzo di checkpoint non quiescente):

flush-nel-commit le pagine modificate da una transazione vengono scritte su disco (subito) prima del commit della transazione stessa;

flush-all-nel-ckpt nel corso del checkpoint (subito prima della sua conclusione), vengono scritte su disco le pagine modificate da tutte le transazioni;

flush-committed-nel-ckpt nel corso del checkpoint (subito prima della sua conclusione), vengono scritte su disco le pagine modificate dalle transazioni andate in commit.

Nella tabella seguente, indicare per ciascuna tecnica quali approcci alla flush possono o debbono essere utilizzati (indicare il nome sintetico e fornire una breve spiegazione; si noti che potrebbe essere utilizzabile una sola tecnica oppure più di una; in questo secondo caso, indicare e spiegare quale sarebbe preferibile).

1. **"undo-only"**

2. **"undo-redo"**

3. **"redo-only"**

Basi di dati II — 12 maggio 2015 — Compito D

Domanda 3 (15%)
Si consideri una base di dati su cui una applicazione effettua moltissimi inserimenti e aggiornamenti, con operazioni tutte molto semplici ma estremamente numerose. Considerare le due situazioni seguenti:

A. concorrentemente alle operazioni sopra citate vengono eseguite molte interrogazioni (e nessun altro aggiornamento)
B. concorrentemente alle operazioni sopra citate non viene eseguita nessun'altra operazione

Commentare brevemente per ciascuna delle due situazioni quali potrebbero essere i vantaggi e gli svantaggi (con riferimento alle prestazioni in termini di tempo di risposta sia per la normale operatività sia in caso di guasto e conseguente necessità di recovery) delle seguenti tre scelte per l'applicazione che esegue inserimenti e aggiornamenti:

i. riunire le operazioni in transazioni di medie dimensioni (alcune decine di operazioni per ciascuna)
ii. eseguire ciascuna operazione in una transazione separata
iii. riunire tutte le operazioni in un'unica transazione

Nota bene: non è detto che esista una soluzione ideale, ciòò che si deve fornire sono riflessioni critiche, che, sinteticamente, illustrino spunti interessanti.

	A	B
i.		
ii.		
iii.		

Basi di dati II — 12 maggio 2015 — Compito D

Domanda 4 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
		begin read(x)
begin read(x)		
	begin read(x)	
x = x + 10 write(x)		
commit		
	x = x + 20	
		read(x)
	write(x)	
	commit	
		commit

Considerare uno scheduler con controllo di concorrenza basato su **2PL** e livello di isolamento **SERIALIZABLE** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 400. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3

Si verificano anomalie?

Indicare brevemente che cosa succede se invece la transazione sul client 3 ha livello di isolamento READ COMMITTED.

Basi di dati II — 12 maggio 2015 — Compito D

Domanda 5 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
begin		
read(x)		
	begin	
	read(x)	
x = x + 10		
write(x)		
		begin
		read(x)
	x = x + 20	
	write(x)	
	commit	
commit		
		read(x)
		commit

Considerare uno scheduler con controllo di concorrenza basato su **Multiversioni** e livello di isolamento **READ COMMITTED** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 400. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — 12 maggio 2015 — Compito D

Domanda 6 (20%)
Considerare un sistema distribuito su cui vengono eseguite due transazioni che coinvolgono tre nodi, un coordinatore C (lo stesso per entrambe le transazioni) e due partecipanti P1 e P2. Dopo la richiesta del coordinatore C di `prepare` (abbreviata con `prep`) per la prima transazione, i due partecipanti ricevono e rispondono correttamente, e uno dei due, P2, va in crash subito dopo aver risposto. Il coordinatore riceve le risposte, prende la decisione, invia i relativi messaggi (questi passi **non** sono stati indicati sotto e vanno quindi scritti) e subito dopo va anch'esso in crash (quindi senza fare in tempo a ricevere le conferme). Viceversa, per la seconda transazione, il coordinatore invia il messaggio di `prepare` ma non fa in tempo a ricevere le risposte. Indicare, nello schema sottostante, una possibile sequenza di scritture sui log e invio di messaggi (che includa anche i passi sopra illustrati), supponendo che entrambi i nodi siano ripristinati abbastanza presto (ma che il coordinatore perda alcuni messaggi di risposta inviati ad esso a seguito del commit). Per i messaggi si usi la notazione *tipo(transaz)*→*destinatari* (come nell'esempio: `prepare(T1)`→`P1,P2`). Supporre che nel log del coordinatore si scrivano solo i record di `prepare`, `commit` e `complete`, con i messaggi gestiti invece in memoria. Indicare ragionevoli istanti per i timeout, che permettano di concludere entrambe le transazioni.

Nodo C		Nodo P1		Nodo P2	
Log	Messaggi	Log	Messaggi	Log	Messaggi
`prep(T1,P1,P2)`	`prep(T1)`→`P1,P2`				
					crash
`prep(T2,P1,P2)`	`prep(T2)`→`P1,P2`				
	crash				
	restart				
					restart

Basi di dati II — Prova parziale — 12 maggio 2015 — Compito A

Cenni sulle soluzioni
(solo Compito A, le varianti del testo sono in rosso)

Tempo a disposizione: un'ora e trenta minuti.
Si suggerisce di scrivere prima una brutta copia, per indicare poi negli spazi le risposte e brevi giustificazioni.

Cognome _____ **Nome** _____ **Matricola** _____

Domanda 1 (20%)
Considerare una tabella T appena creata (e quindi vuota), con le seguenti ipotesi

- T è definita su due campi, A di lunghezza $a = 4$ byte e B di lunghezza $b = 34$ byte, senza vincoli espliciti di chiave (e quindi le operazioni si possono fare senza verifiche particolari);
- la struttura fisica utilizzata per T è heap, senza indici, con una memorizzazione a lunghezza fissa (in cui supponiamo che, oltre ai byte necessari per i campi, ne servano 2 ulteriori per la memorizzazione) e in cui si marcano come liberi gli spazi dei record eliminati, riutilizzandoli per successivi inserimenti (come avviene in SimpleDB);
- il sistema utilizza blocchi di dimensione $D = 4$ Kbyte (approssimabili a 4000).

In tale contesto, supporre che vengano eseguite le seguenti operazioni

1. inserimento di $L = 200.000$ ennuple
2. eliminazione di $L/4 = 50.000$ ennuple (sulla base di una condizione verificabile durante la scansione)
3. dopo la conclusione e la chiusura della scansione precedente, inserimento di altre $L/4 = 50.000$ ennuple

Rispondere alle domande seguenti, indicando formule e valori numerici:

Fattore di blocco f per la relazione T:
$$f = D/(a + b + 2) = 4000/40 = 100$$

Numero blocchi occupati da T dopo la prima serie di inserimenti (punto 1):
$$L/f = 200.000/100 = 2000$$

Numero blocchi occupati da T dopo le eliminazioni di cui al punto 2:
$$L/f = 200.000/100 = 2000$$
(i blocchi restano gli stessi, con lo spazio libero marcato e non utilizzato)

Numero blocchi occupati da T dopo la seconda serie di inserimenti (punto 3):
$$L/f = 200.000/100 = 2000$$
(lo spazio libero viene riutilizzato)

Numero di scritture in memoria secondaria (per le pagine dei dati e quelle del log) per la prima serie di inserimenti, supponendo che, in questo caso, i record di log abbiano una lunghezza pari a circa il doppio di quella dei record del file, con riferimento ad un programma che utilizzi un'unica transazione per tutti gli inserimenti (e supponendo che non vi siano altre transazioni attive)

- numero di scritture di pagine di log:

  dovrebbe essere possibile scrivere tutto il log al momento del commit e quindi (poiché i record sono grandi il doppio di quelli del file): $2 \times L/f = 4000$

- numero di scritture di pagine di dati:

  in ogni caso si dovrebbe avere $L/f = 2000$, una per blocco

Come nel caso precedente, ma con riferimento ad un programma che utilizzi una transazione separata per ciascun inserimento

- numero di scritture di pagine di log:

  almeno $L = 200.000$, una per transazione

- numero di scritture di pagine di dati:

  con una strategia undo-only, una per transazione, $L = 200.000$; altrimenti, probabilmente di meno, anche solo $L/f = 2000$, una per blocco

Basi di dati II — 12 maggio 2015 — Compito A

Domanda 2 (15%)
Come noto, esistono tecniche leggermente diverse per il recovery, che prevedono solo redo (**"redo-only"**), solo undo (**"undo-only"**) oppure tanto undo quanto redo (**"undo-redo"**). Le tecniche differiscono, oltre che nella procedura di recovery, anche nella modalità di esecuzione delle scritture effettive su disco (operazioni di "flush" delle pagine di buffer), che possono essere le seguenti (con riferimento ad un utilizzo di checkpoint non quiescente):

flush-nel-commit le pagine modificate da una transazione vengono scritte su disco (subito) prima del commit della transazione stessa;

flush-all-nel-ckpt nel corso del checkpoint (subito prima della sua conclusione), vengono scritte su disco le pagine modificate da tutte le transazioni;

flush-committed-nel-ckpt nel corso del checkpoint (subito prima della sua conclusione), vengono scritte su disco le pagine modificate dalle transazioni andate in commit.

Nella tabella seguente, indicare per ciascuna tecnica quali approcci alla flush possono o debbono essere utilizzati (indicare il nome sintetico e fornire una breve spiegazione; si noti che potrebbe essere utilizzabile una sola tecnica oppure più di una; in questo secondo caso, indicare e spiegare quale sarebbe preferibile).

1. **"undo-redo"**

flush-all-nel-ckpt: serve per garantire che le transazioni concluse al checkpoint non abbiano bisogno di essere ribadite; può portare ad undo per transazioni non concluse, ma è più semplice da gestire delle altre due, che pure sarebbero corrette

2. **"redo-only"**

flush-committed-nel-ckpt: serve a garantire che le transazioni andate in commit prima del checkpoint non siano da rifare; non scrive altro su disco e quindi non c'è necessità di undo. Non va bene la **flush-nel-commit** perché ci potrebbe essere un crash fra la flush e il commit e allora servirebbe l'undo. Analogamente, non va bene la **flush-all-nel-ckpt** perché si scriverebbero su disco dati non confermati da commit e quindi portebbero servire redo.

3. **"undo-only"**

flush-nel-commit, in quanto garantisce che non ci sia mai bisogno di redo (i dati sono su disco al momento del commit); non si può aspettare il checkpoint, perché se ci fosse un crash prima di esso allora servirebbero redo

Basi di dati II — 12 maggio 2015 — Compito A

Domanda 3 (15%)
Si consideri una base di dati su cui una applicazione effettua moltissimi inserimenti e aggiornamenti, con operazioni tutte molto semplici ma estremamente numerose. Considerare le due situazioni seguenti:

A. concorrentemente alle operazioni sopra citate non viene eseguita nessun'altra operazione
B. concorrentemente alle operazioni sopra citate vengono eseguite molte interrogazioni (e nessun altro aggiornamento)

Commentare brevemente per ciascuna delle due situazioni quali potrebbero essere i vantaggi e gli svantaggi (con riferimento alle prestazioni in termini di tempo di risposta sia per la normale operatività sia in caso di guasto e conseguente necessità di recovery) delle seguenti tre scelte per l'applicazione che esegue inserimenti e aggiornamenti:

i. eseguire ciascuna operazione in una transazione separata
ii. riunire le operazioni in transazioni di medie dimensioni (alcune decine di operazioni per ciascuna)
iii. riunire tutte le operazioni in un'unica transazione

Nota bene: non è detto che esista una soluzione ideale, ciòò che si deve fornire sono riflessioni critiche, che, sinteticamente, illustrino spunti interessanti.

Commenti sulle soluzioni
Considerazioni generali, da combinare poi per rispondere alle domande:

- rispetto alla concorrenza (e quindi nel caso in cui ci siano di esecuzione le interrogazioni, che sono molte), la singola transazione molto probabilmente penalizza, perché blocca e/o viene bloccata; la soluzione estrema opposta è molto più efficiente, quella intermedia è probabilmente pure efficiente (se la base di dati è grande, bloccherà solo una piccola parte dei dati)
- più sono le transazioni, maggiore è l'onere per la gestione della affidabilità (con inizio e fine di transazioni e scritture, anche forzate, sul log)
- in caso di guasto, la soluzione con transazione unica porta all'annullamento di tutte le azioni svolte (il commit è alla fine e quindi un eventuale guasto sarà prima di esso), con la necessità quindi di ripetere il lavoro

Basi di dati II — 12 maggio 2015 — Compito A

Domanda 4 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
		begin
		read(x)
begin		
read(x)		
	begin	
	read(x)	
x = x + 10		
write(x)		
commit		
	x = x + 20	
		read(x)
	write(x)	
	commit	
		commit

Considerare uno scheduler con controllo di concorrenza basato su **2PL** e livello di isolamento **READ COMMITTED** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 400. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3
		begin
		read(x) *legge 400*
begin		
read(x) *legge 400*		
	begin	
	read(x) *legge 400*	
x = x + 10		
write(x) *scrive 410*		
commit		
	x = x + 20	
		read(x) *legge 410*
	write(x) *scrive 420*	
	commit	
		commit

Si verificano anomalie?
Perdita di aggiornamento fra il client 1 e il client 2 e lettura inconsistente per il client 3

Basi di dati II — 12 maggio 2015 — Compito A

Domanda 5 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
begin		
read(x)		
	begin	
	read(x)	
x = x + 10		
write(x)		
		begin
		read(x)
	x = x + 20	
	write(x)	
	commit	
commit		
		read(x)
		commit

Considerare uno scheduler con controllo di concorrenza basato su **Multiversioni** e livello di isolamento **SERIALIZABLE** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 400. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3
begin		
read(x) *legge 400*		
	begin	
	read(x) *legge 400*	
x = x + 10		
write(x) *scrive 410*		
		begin
		read(x) *legge 400*
	x = x + 20	
	xlock(x) *attesa*	
commit		
	abort	
	begin	
	read(x) *legge 410*	
	write(x) *scrive 430*	
	commit	
		read(x) *legge 400*
		commit

Si verificano anomalie?
Nessuna

Indicare brevemente che cosa succede se invece la transazione sul client 3 ha livello di isolamento READ COMMITTED.
Risposta
Il cliente 3 legge valori in commit al momento e quindi la seconda lettura legge un valore diverso rispetto alla prima

Basi di dati II — 12 maggio 2015 — Compito A

Domanda 6 (20%)
Considerare un sistema distribuito su cui vengono eseguite due transazioni che coinvolgono tre nodi, un coordinatore M (lo stesso per entrambe le transazioni) e due partecipanti R1 e R2. Dopo la richiesta del coordinatore M di `prepare` (abbreviata con `prep`) per la prima transazione, i due partecipanti ricevono e rispondono correttamente, e uno dei due, R2, va in crash subito dopo aver risposto. Il coordinatore riceve le risposte, prende la decisione, invia i relativi messaggi (questi passi **non** sono stati indicati sotto e vanno quindi scritti) e subito dopo va anch'esso in crash (quindi senza fare in tempo a ricevere le conferme). Viceversa, per la seconda transazione, il coordinatore invia il messaggio di `prepare` ma non fa in tempo a ricevere le risposte. Indicare, nello schema sottostante, una possibile sequenza di scritture sui log e invio di messaggi (che includa anche i passi sopra illustrati), supponendo che entrambi i nodi siano ripristinati abbastanza presto (ma che il coordinatore perda alcuni messaggi di risposta inviati ad esso a seguito del commit). Per i messaggi si usi la notazione *tipo(transaz)*→*destinatari* (come nell'esempio: `prepare(T1)`→`R1,R2`). Supporre che nel log del coordinatore si scrivano solo i record di `prepare`, `commit` e `complete`, con i messaggi gestiti invece in memoria. Indicare ragionevoli istanti per i timeout, che permettano di concludere entrambe le transazioni.

Nodo M		Nodo R1		Nodo R2	
Log	Messaggi	Log	Messaggi	Log	Messaggi
`prep(T1,R1,R2)`	`prep(T1)`→`R1,R2`				
		`ready(T1)`		`ready(T1)`	
			`ready(T1)`→`M`		`ready(T1)`→`M`
				crash	
`commit(T1)`					
	`commit(T1)`→`R1,R2`				
		`commit(T1)`			
			`ack(T1)`→`M`		
`prep(T2,R1,R2)`					
	`prep(T2)`→`R1,R2`				
		`ready(T2)`			
			`ready(T2)`→`M`		
	crash				
	restart				
	`commit(T1)`→`R1,R2`				
`abort(T2)`					
	`abort(T2)`→`R1,R2`				
			`ack(T1)`→`M`		
		`abort(T2)`			
			`ack(T2)`→`M`		
				restart	
	`commit(T1)`→`R1`				
	`abort(T2)`→`R1`				
				`commit(T1)`	
					`ack(T1)`→`M`
				`abort(T2)`	
					`ack(T2)`→`M`
`complete(T1)`					
`complete(T2)`					